Unsupervised learning for local structure detection in system with hard spheres

Georgios Smyridis, Rimo Sarkar

1 Introduction

A significant challenge to overcome when studying the self-assembly products of colloidal systems, is phase detection. For systems of which the basic phases are known, one can construct order parameters that describe the local structure of single particles, and thus detect which expected phase the particle is in. However, in many cases the expected phases of the system are not known, and thus, the selection of an appropriate order parameter is complicated. In such a context, unsupervised (machine) learning, being effective in detecting patterns and structures in large datasets, proves itself very valuable.

In this report we are going to provide an example of phase detection for a system of hard spheres, employing unsupervised learning techniques based on bond order parameters. In Section 2 we describe the algorithm that we are going to use for local environment classification, providing a concise description for each component technique. In section 3, we present our results, and compare with the ones provided in the literature [1].

2 Theory

The overall process of phase detection in our algorithm consists of three steps: First, we describe each particle's local environment using bond orientational order parameters. There are ample of such parameters and they can be organised in a high dimensional vector space. Not all of them are relevant, and to extract the relevant information, we use dimensionality reduction with neural-network-based autoencoders. In that lower dimensional subspace, hopefully, the particles that are in the same phase will be grouped together with a clustering algorithm (Gaussian Mixture Models). After we have clustered the lower-dimensional representation of our data, we go back to the original vector space that our data live and plot the distribution of BOPs for each cluster. What we look for each averaged BOPs that have well separated distributions for the different clusters. These degrees of freedom are the ones that predict the phase of the system. In the final step we rank these predictive parameters by order of importance using contribution analysis (input perturbation and improves stepwise function).

2.1 Local environment description with bond-order-paraeters

To characterise the local environment of each particle, we use the averaged bond order parameters (BOPs) [2]. For any given particle i we define the complex quantities

$$q_{lm}(i) = \frac{1}{N_b(i)} \sum_{j \in \mathcal{N}_b(i)} Y_l^m(\boldsymbol{r}_{ij}) \tag{1}$$

where $Y_l^m(\boldsymbol{r}_{ij})$ are the spherical harmonics of order l, with m an integer that runs from $m = -l$ to $m = l$. Additionally, $\boldsymbol{r}_{ij}$ is the vector from particle i to particle j, and $\mathcal{N}_b(i)$ is the set of nearest neighbours of particle i, which we define later. Note that $|\mathcal{N}_b(i)| = N_b(i)$. Then, we define the

average $\bar{q}_{lm}(i)$ as

$$\bar{q}_{lm}(i) = \frac{1}{N_b(i)+1} \sum_{k \in \{i\} \cup \mathcal{N}_b(i)} q_{lm}(k) \tag{2}$$

where the sum runs over all nearest neighbours of particle i as well as particle i itself. Averaging over the nearest neighbour values of q_{lm} results in effectively in also taking next-nearest neighbours into account. Finally, we define rotationally invariant BOPs as

$$\bar{q}_l(i) = \sqrt{\frac{4\pi}{2l+1} \sum_{m=-l}^{l} |\bar{q}_{lm}(i)|^2} \tag{3}$$

which, depending on the choice of l, are sensitive to different crystal symmetries.

The optimal set of BOPs to be considered strongly depends on the structures one wishes to distinguish. Since our method is meant to be applied to systems for which such prior knowledge is missing, in order to describe the local environment of one particle, we evaluate several $\bar{q}_l$ with l ranging from 1 to 8. Therefore, when considering one component systems, our description of the local environment of particle i is encoded into an 8-dimensional vector

$$\boldsymbol{Q}(i) = (\{\bar{q}_l\}), \tag{4}$$

with l integer from 1 to 8.

2.1.1 Nearest Neighbours

When calculating the BOPs, one is confronted with the task of determining the nearest neighbours of a particle. However, in practice, there is no unique definition of nearest neighbours. The definitions have opted to make use of, is a recently introduced nearest-neighbour criterion, called solid angle nearest neighbours SANNs [3].

In this approach, an effective individual cutoff is found for every particle in the system based on its local environment. More concretely, consider a system where we have a particle i located at position $\boldsymbol{r}_i$ surrounded by particles $\{j\}$. The fixed-distance cutoff defines the set of nearest neighbours of particle i to be all the particles $\{j\}$ with distance to i smaller than some threshold R. i.e.

$$\mathcal{N}_b(i) = \{n : n \in \{j\} \wedge r_{in} \leq R\} \tag{5}$$

As mentioned above, the SANN algorithm determines an *individual* cutoff distance $R_i^{(m)}$ for each particle i, which we call the *shell radius*. This radius depends solely on the local environment of each particle i and includes its m nearest neighbours. For the computation of $R_i^{(m)}$ SANN uses a purely geometrical construction and is thus parameter-free and scale-free.

Now, we move on to the calculation of the individual cut-off distance $R_i^{(m)}$. First, we order the particles $\{j\}$ surrounding i such that $r_{i,j} \leq r_{i,j+1}$ for all j. This relates the shell distance $R_i^{(m)}$ and the number of nearest neighbours m in the following manner:

$$r_{i,m} \leq R_i^{(m)} < r_{i,m+1}. \tag{6}$$

Then, starting with the particle closest to i we associate with each potential neighbour j an angle $\theta_{i,j}$ based on the distance between the particles $r_{i,j} = |\boldsymbol{r}_i - \boldsymbol{r}_j|$ and the yet undetermined shell radius $R_i^{(m)}$

SANN defines the neighborhood of a particle i to consist of the closest m particles $\{j\}$ such that the sum of their solid angles associates with $\theta_{i,j}$ equals to 4π, i.e.,

$$4\pi = \sum_{j=1}^{m} 2\pi\big(1 - \cos\theta_{i,j}\big) = \sum_{j=1}^{m} 2\pi \left(1 - \frac{r_{i,j}}{R_i^{(m)}}\right) \tag{7}$$

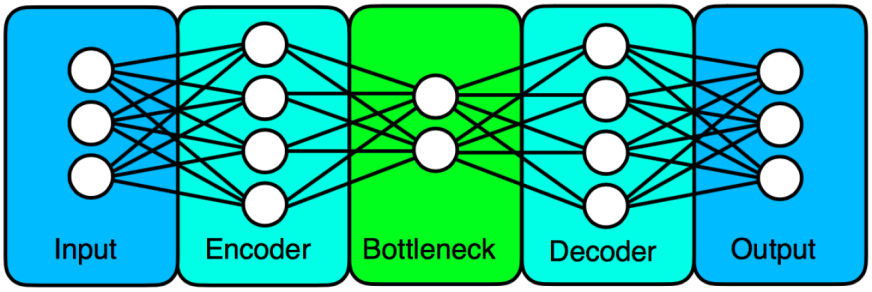


Figure 1: Architecture of a neural-network based autoencoder. The autoencoder finds a low-dimensional representation of the input, from which the decoder reconstructs an approximation of the input as output.

Combining (6) and (7) leads to a condition for the determination of the neighbour shell radius,

$$R_i^{(m)} = \frac{\sum_{j=1}^{m} r_{i,j}}{m-2} < r_{i,m+1}, \tag{8}$$

where $R_i^{(m)}$ refers to the shell radius containing m particles. To solve this inequality, we start with the smallest number of neighbours capable of satisfying it, $m = 3$, and increase m iteratively. During each iteration, we evaluate (8) and the smallest m that satisfies the equation yieds the number of neighbours $N_b(i)$ with $R_i^{(m)}$ the corresponding neighbour shell radius.

This method is not inherently symmetric, i.e. j might be a neighbour of i while i is not a neighbour of j. However, symmetry can be enforced by either adding j to the neighbours of i or removing i from the neighbours of j. We choose to enforce symmetry with the latter approach.

2.2 Dimensionality Reduction: Autoencoders

Autoencoders [4] are a type of artificial neural network (ANN) architecture that are primarily used for unsupervised learning and dimensionality reduction. They are designed to learn efficient representations of input data by compressing the input into a lower-dimensional latent space and then reconstructing the original input from this compressed representation.

In the present context, we employ feedforward and fully connected autoencoders like in the figure. The number of input and output nodes, d is specified by the dimensionality of the input vector $\boldsymbol{Q}(i) \in \mathbb{R}^d$, which are approximately reconstructed by the network in the output layer

$$\hat{\boldsymbol{Q}}(i) = \boldsymbol{\Theta}_{\text{autoencode}}(\boldsymbol{Q}(i)) = (\boldsymbol{\Theta}_{\text{decode}} \circ \boldsymbol{\Theta}_{\text{encode}})(\boldsymbol{Q}(i)) \in \mathbb{R}^d. \tag{9}$$

The bottleneck layer contains the low-dimensional projection to be learned by the encoder, $\boldsymbol{Y}(i) = \boldsymbol{\Theta}_{\text{encode}}(\boldsymbol{Q}(i)) \in \mathbb{R}^c$, whose dimensionality is controlled by the number of bottleneck nodes, $c < d$.

Each node k in layer i operates on its input, $x_k^{(i)}$, via an activation function to produce an output $y_k^{(i)} = f^{(i)}(x_k^{(i)})$. The input to each node in a fully connected, feedforward architecture (without bias) is the weighted sum of the output of the previous layer

$$x_k^{(i)} = \sum_j w_{jk}^{(i-1)} y_j^{(i-1)} = \boldsymbol{W}^{(i-1)} \cdot \boldsymbol{y}^{(i-1)}, \tag{10}$$

where here $w_{jk}^{(i-1)}$ is the weight of the connection between node j in layer $(i-1)$ and node k in node i. Nonlinearity is achieved by providing both the encoder and the decoder with a fully-connected

hidden layer with a nonlinear activation function. Here, we set the number of nodes in the hidden layer to $10d$ and use a hyperbolic tangent as the activation function. For the bottleneck and output layers, instead, a linear activation function is used.

The training of the network amounts to adjusting the weights of the network so as to optimise the reconstruction error of the input dataset

$$E\big(\boldsymbol{W}, \{\boldsymbol{Q}(i)\}\big) = \sum_i \big|\boldsymbol{Q}(i) - \hat{\boldsymbol{Q}}(i)\big|^2 + \Gamma\big(\boldsymbol{W}\big), \tag{11}$$

where $\Gamma\big(\boldsymbol{W}\big)$ is a regularisation term in order to control the magnitude of the network weights during training and thus prevent overfitting. In our case, we choose L2 regularisation, which means that $\Gamma(\boldsymbol{W}) = \lambda \sum_{i,j,k} \big(w_{jk}^{(i)}\big)^2$. The optimisation is conducted using mini-batch stochastic gradient descent with momentum.

2.3 Contribution Analysis

ANNs have the capacity to predict output variable ,but they are often considered black boxed, in the sense that the mechanisms that occur within the network are often ignored. Various authors have explored this problem and proposed algorithms to illustrate the role of variables in ANN models. Most works, however, focus on methods that eliminate irrelevant input and reduce the size of the network.

Even though good prediction and eliminating redundancy is desirable, knowing what contribution each variable makes is of prime importance as well. Contribution analysis uses methods to determine the influence of each input variable and its contribution to the output. They are not, therefore, pruning methods but procedures to estimate the relative contribution of each input variable. The methods that will interest us are the "input perturbation" and the "improved stepwise" methods [6]. As we will see, both techniques require a single trained model, avoid having to repeat the training multiple times.

2.3.1 Input Perturbation Method

This input perturbation method aims to assess the effect of small perturbations in each input of the neural network to its output. The algorithm adjusts the input values of one variable, by adding a perturbation, $x_i \to x_i + \delta$ with $\delta/x_i \in [0, 0.5]$, while keeping the rest fixed. The effect of these input perturbations is reflected in the cost function, say MSE. In fact, the MSE of the neural network output is expected to increase as a larger amount of noise is added to the selected input variable. Wen can thus rank the importance of each input node based on the increase in MSE, the largest such change corresponding to the input node of highest importance.

2.3.2 Improved Stepwise Method

Stepwise selection is an iterative process that starts with an empty model and incrementally adds or removes features based on certain criteria. The two main variations of stepwise selection are forward selection and backward elimination, the one being the mirror image of the other. In our work, we assess the relative importance of the input nodes using backwards elimination.

In this approach, the algorithm starts with a model containing all the features and removes one feature at a time, replacing all its values with its mean over the whole dataset. At each step, the algorithm evaluates the performance of the model after removing a feature and selects the one whose removal has the least impact on the model's performance based on the magnitude of the cost function, MSE. The process continues until removing any additional feature significantly degrades the model's performance.

2.3.3 Relative Importance

Both methods evaluate the importance of each node by its effect on the cost function, MSE. We quantify the relative importance of the k-th input node with

$$RI_k = \frac{\Delta E_k}{\sum_{j=1}^{d} \delta E_j}, \tag{12}$$

with ΔE_j being the change in MSE caused by the change applied to the input node j, and the sum in the denominator is taken over all the input nodes.

2.4 Clustering: Gaussian Mixture Models

Clustering is the task of examining a dataset, and trying to find similar instances and assigning them to clusters, or groups of similar instances. Clustering can be though of as a classification task with the core difference that it is unsupervised as opposed to supervised. The algorithm we use for the clustering in this work is the Gaussian Mixture Models (GMMs)

2.5 Gaussian Mixtures

GMMs are a type of generative model utilized to represent intricate data distributions and are especially valuable for performing data clustering. By being generative, GMMs aims to understand the underlying probability distribution of the input data. The fundamental assumption is that data points within each cluster originate from a multivariate Gaussian distribution.

GMMs merge multiple Gaussian distributions to construct a mixture model. Each Gaussian distribution corresponds to a distinct component within the data. The mixture model defines the proportions or weights assigned to each component, denoting their respective contributions to the overall distribution.

The objective of GMMs training involves estimating the model parameters such as means, covariance matrices, and weights, which best capture the observed data. This estimation is commonly accomplished using the Expectation Maximization (EM) algorithm.

2.5.1 Finding the number of clusters

The most straightforward way to cluster data using GMMs involves treating each mixture component as a distinct cluster and assigning each observation to the component that has the highest posterior probability. However, this approach works well only when the clusters are genuinely generated from a combination of separate multivariate normal distributions. In reality, the clusters underlying our data often deviate significantly from Gaussian distribution in space. Consequently, a single cluster in the data may be identified as two or more mixture components if its distribution is better approximated by a mixture of Gaussians rather than a single Gaussian function. This implies that the number of clusters in the data may differ from the number of components by minimizing the Bayesian Information Criterion (BIC) [7].

[8] argues that the goal of selecting the number of mixture components for estimating the underlying probability density is well met by BIC, and thus it is used to select the number of components in the mixture model. They then propose a sequence of possible solution by a hierarchical combination of the components identified by BIC. The dicision about which components to combine is based on minimising the entropy of the resulting clustering define as

$$S_K = -\sum_{i=0}^{N}\sum_{j=1}^{K} p_{ij} \ln(p_{ij}) \tag{13}$$

where N is the number of observations and K is the number of clusters. Practically, the optimal number of clusters is found by looking at the existense of an elbow in the entropy S_K as a function of K.

3 Results

To test this methodology we apply it for a simulated system of 1372 hard spheres with diameter d, in the NVT ensemble. The initial configuration for each simulation is an FCC crystal, with lattice spacing $4 * d$, and then we set the desired phase by setting the appropriate number density. More specifically, we simulated two FCC crystals and two fluid phases with different number densities, provided in Table 1. For each different system we took three snapshots over time, that is, we saved the positions of all particles. We end up in total with 12 snapshots l, and thus a dataset of 16464 points.

Phase	Number density ρd^3
FCC1	1.05
FCC2	1.01
FLUID1	0.86
FLUID2	0.80

Table 1: Number of density ρd^3 for each phase of the system we simulated.

For each snapshot, we assign to each particle an 8-dimensional vector $\boldsymbol{Q}_l(i)$, where i is the particles, and l is the snapshot. The details of the construction are given in Section 2.1.

3.0.1 Reducing the dimensionality with the autoencoder

Next, we move on to reduce dimensionality with the autoencoder. It important first to determine the number of bottleneck nodes. This is done by training the autoencoder for number of nodes ranging from one to eight and plot the loss function with the number of bottleneck nodes in Figure 2a. We observe that an "elbow" [5] is formed for number of bottleneck nodes $c = 2$, even though admittedly the elbow is not clear.

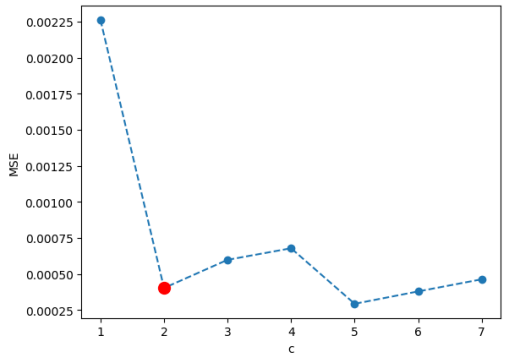
3.0.2 Clustering

Subsequently, we train the autoencoder for $c = 2$ and keep the lower dimensional representation of our data, the code. With this output we train Gaussian mixture models for different number of components, and we plot the BIC in Figure 2b. The minimum value is acquired for number of Gaussian components $N_G = 5$. The next step is to determine the number of different clusters, by merging each time two components and minimising the entropy of the resulting clustering. The plot of this entropy with the number of clusters is given in Figure 2c, and an elbow is present for number of clusters $K = 2$. The GMM now clusters the data in the lower dimensional represention into two distinct clusters (see Figure 2d) that likely represent the two distinct phases, namely, FCC and fluid.

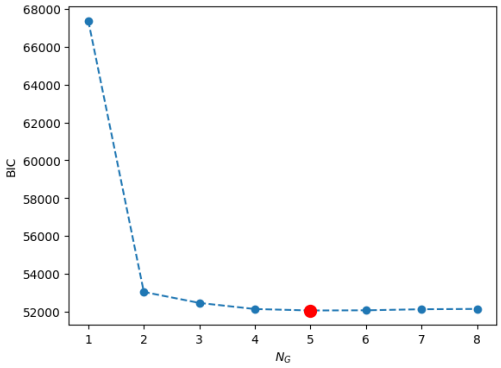
After we have clustered the 2-dimensional representation of our data, we want to go back to the original 8-dimensional vector space and look at the effect of the clustering there. Specifically, we plot the distributions of the averaged BOPs for each cluster in Figure 3. We observe that only for $\bar{q}_4$, $\bar{q}_6$ and $\bar{q}_8$ the distributions are relatively well separated. Henceforth, we project our original data onto the planes $\bar{q}_l - \bar{q}_k$ for the aforementioned averaged BOPs and we observe the separation there (Figure 4).

3.0.3 Relative importance

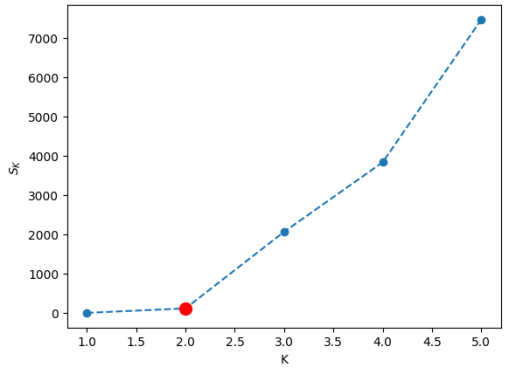
Finally, after we have found that the averaged BOPs that have predictive power for the resulting clustering, we want to rank them by predictive performance, by carrying out contribution analysis. The results are given in Figure



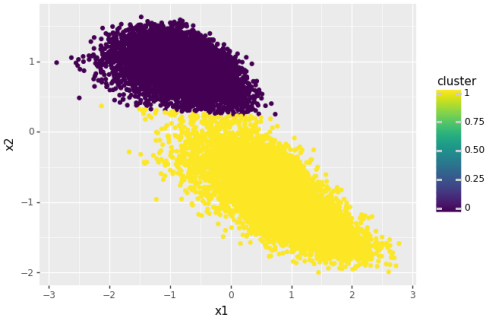
(a) MSE of the autoencoder as a function of the number of bottleneck nodes. We notice the presence of an "elbow" at $c = 2$, highlighted in red.



(b) BIC of GMMs as a function of the number of its components. The minimum for $N_G = 5$ is highlighted in red.



(c) Entropy of the resulting clustering made by GMMs as a function of the number of clusters. We detect an elbow for $K = 2$, highlighted in red.



(d) Projection of the vector $\boldsymbol{Q}(i)$ onto the 2-dimensional space found by the encoder. Colors represent the distinct environments identified by the clustering.

Figure 2: Analysis snapshot from

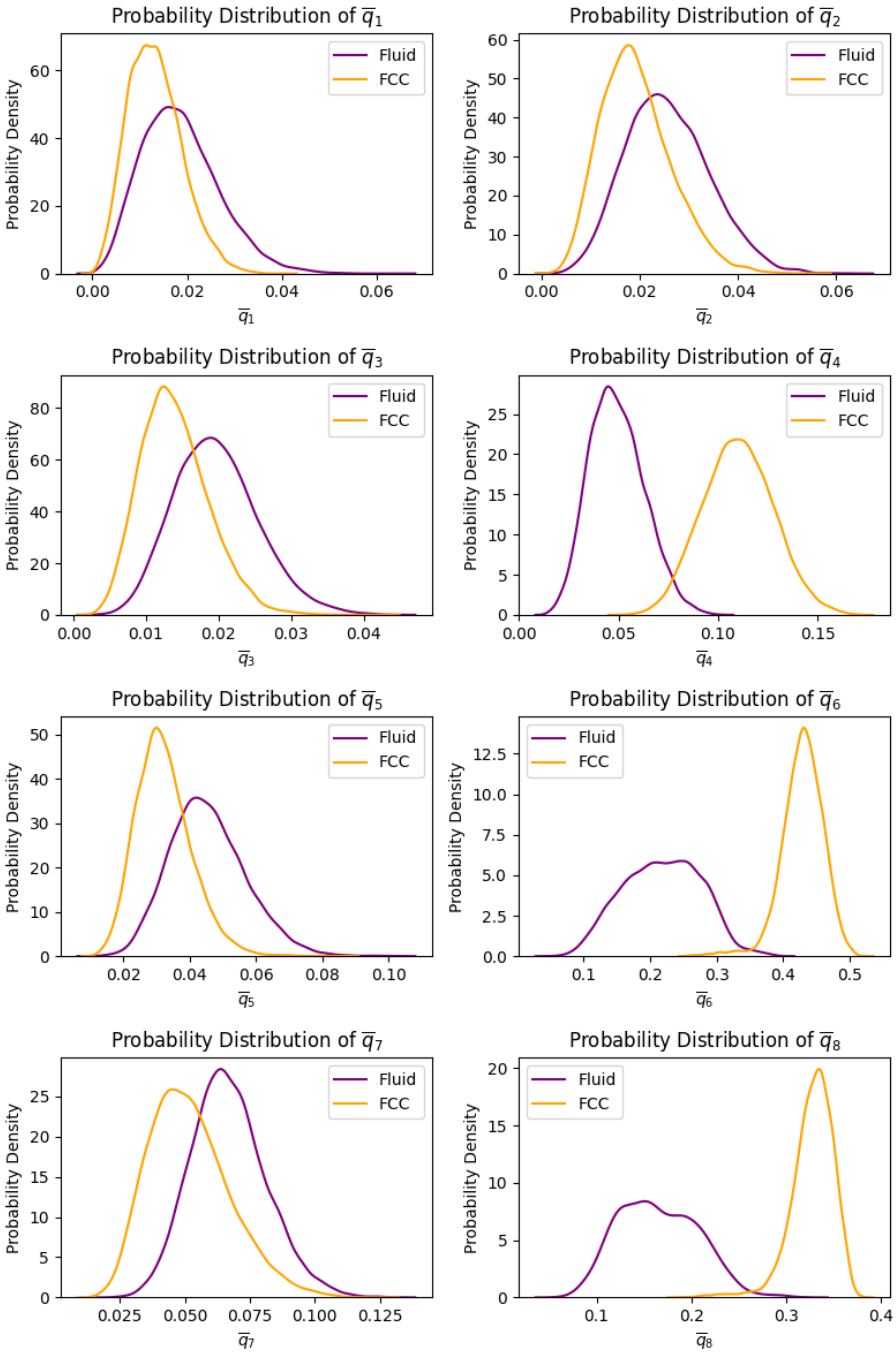


Figure 3: Probability distributions of averaged BOPs for each cluster. We observe that they are well separated for $\bar{q}_4$, $\bar{q}_6$ and $\bar{q}_8$

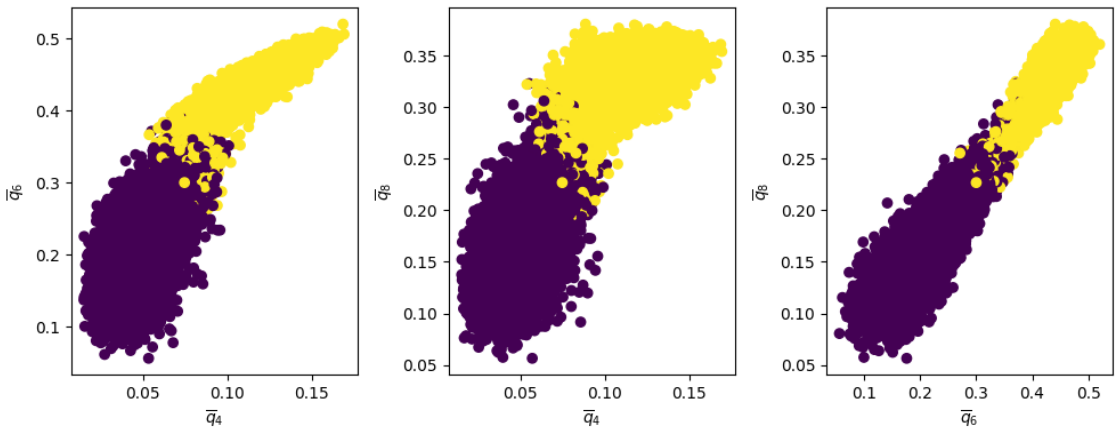


Figure 4: Projections of clustered data in the original 8-dimensional space.

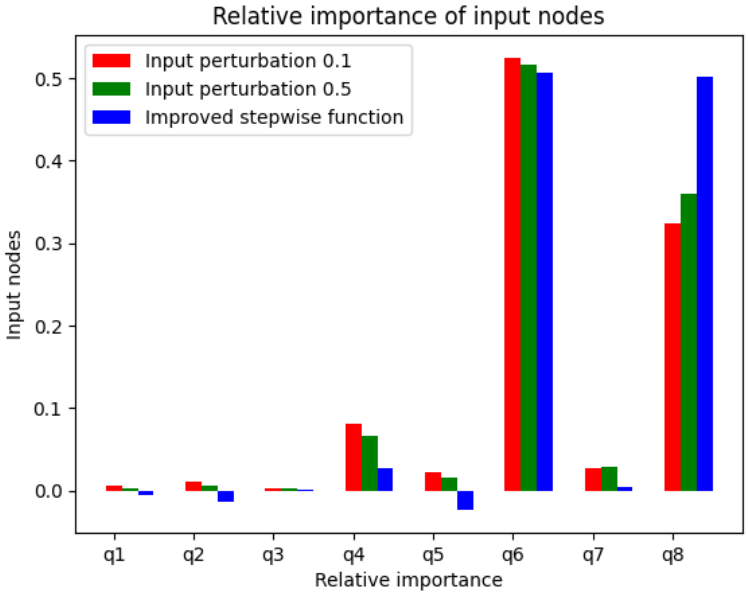


Figure 5: Relative importance of each averaged BOP.

4 Conclusions & Outlook

The results we got in Section 3 point to the fact that indeed unsupervised learning can be used to detect local phase of materials. For a system with hard spheres, the averaged BOPs that predict the local phase of each particles, in order of importance are $\bar{q}_8$, $\bar{q}_6$ and $\bar{q}_4$. Our results seem to agree sufficiently with the ones provided in [1]. For future work, I would like to apply the methodology in colloidal systems and detect the self assembly products that are formed, as is done in [1].

References

[1] Emanuele Boattini, Morjolein Dijkstra, Laura Filion (2019): Unsupervised learning for local structure detection in colloidal systems, arXiv:1907.02420v1

[2] J. Steinhardt, D. R. Nelson, and M. Ronchetti, Phys. Rev. B 28, 784 (1983).

[3] J. A. van Meel, L. Filion, C. Valeriani, and D. Frenkel, J. Chem. Phys. 136, 234107 (2012).

[4] C. M. Bishop, Neural Networks for Pattern Recognition (Oxford University Press, Inc., New York, NY, USA, 1995).

[5] S. Salvador and P. Chan, 16th IEEE International Conference on Tools with Artificial Intelligence , 576 (2004).

[6] M. Gevrey, I. Dimopoulos, and S. Lek, Ecol. Model. 160, 249 (2003).

[7] G. Schwarz, Ann. Statist. 6, 461 (1978).

[8] J. Baudry, A. Raftery, G. Celeux, K. Lo, and R. Gottardo, J. Comput. Graph. Stat. 19, 332 (2010).